



Effective Coding With AI

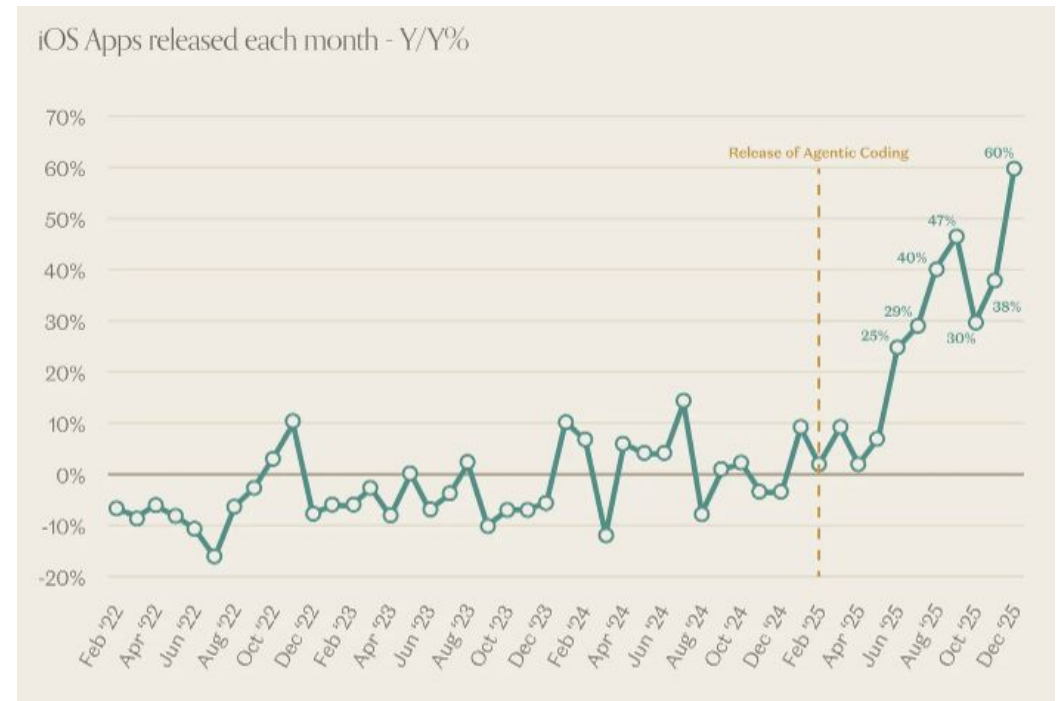
Case Study: Tea Dating Advice & Security Ownership

CMU 15-113 | LECTURE

Rise of Vibe Coded Apps

Since the release of Agentic Coding, the number of app released has jumped

- Many of which are completely AI generated
- Created by the general public
- Would you trust a rando with your data?



Take Tea For Example

The Product

Tea Dating Advice was designed to help women date safely. Providing checks verified by other women.



The app gained popularity...



 **CBS NEWS**



LIVE

CBS News 24/7

MoneyWatch

Forget ChatGPT. Tea is the most downloaded app on Apple's App Store.

But...

in mid-2025, the very app promising safety suffered a catastrophic privacy failure



Watch Live



Subscribe
\$1/wk

Sign In

CULTURE

Tea encouraged its users to spill. Then the app's data got leaked

My ex stalked me, so I joined a 'dating safety' app. Then my address was leaked

AUGUST 2, 2025 1:57 PM ET



Tea app fallout worsens as leaked selfie rating site, online map

By Austin Williams | Published July 28, 2025 5:22pm EDT | Social Media | FOX Local

 NBC NEWS

SUBSCRIBE

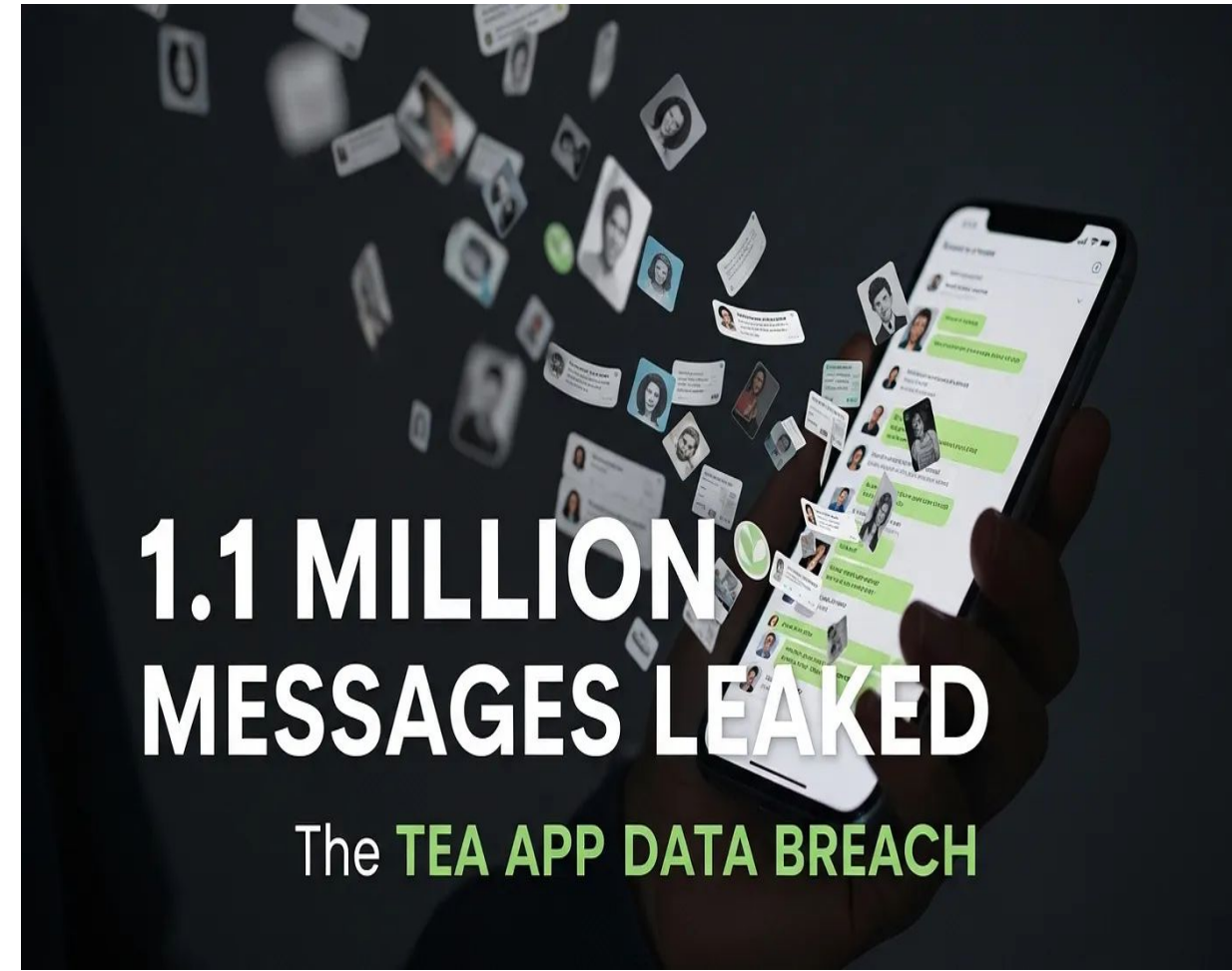


Hackers leak 13,000 user photos and IDs from the Tea app, designed as a women's safe space

The viral app requires new users to take selfies, which it says it deletes after review.

Why it Failed

- Sensitive data was stored in a Cloud database with...
 - No passwords
 - No Encryption
- A simple data scraping script stole it all
- Failure in data retention
 - Soft deletes
- Tea app's API "did not verify the user authentication properly".
 - Insecure Direct Object Reference (IDOR)



Consequences of Failure

10+

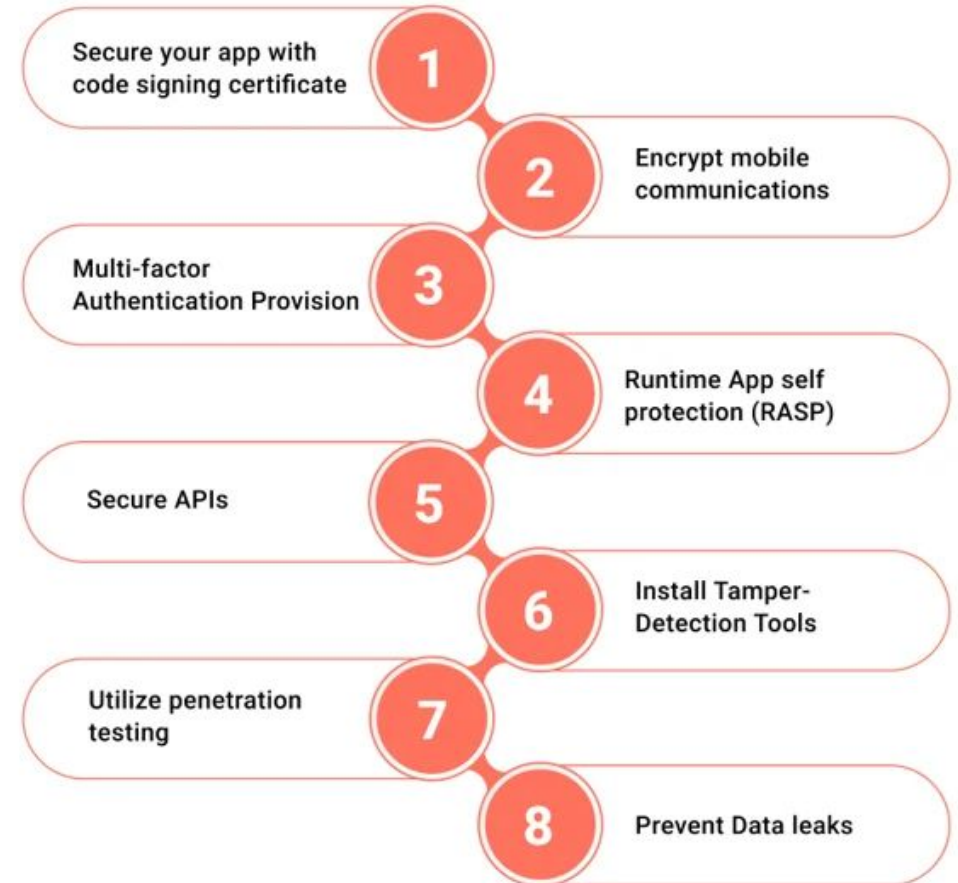
Class Action Lawsuits

Financial Liability

Legal experts suggest the breach could result in **tens of millions of dollars** in damages.

Some Preventative Measures

- Secure cloud Configurations by authentication
 - AWS, Azure, GCP
- Encrypt sensitive data
- Prevent IDOR - authorization checks
- Collect only necessary information
- Delete unnecessary data
- Implement rate limits to brute force attacks
- Input validation to prevent injection attacks



Lecture Poll

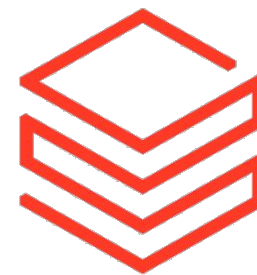


<https://forms.gle/t9anL9J6dT916zMU6>

Multiplayer Snake Game

Similarly to many live demos we've done before in lecture, Databricks conducted a study on "vibe coding".

One of the tests they went with is a multiplayer Snake game.....



databricks

```

def receive_game_state(self):
    while self.running:
        try:
            data = self.client_socket.recv(4096)
            if data:
                game_state = pickle.loads(data)

                # First game state includes player ID
                if 'player_id' in game_state:
                    self.snake.player_id = game_state['player_id']
                    print(f"Assigned player ID: {self.snake.player_id}")

                # Update other snakes and fruits
                if 'snakes' in game_state:
                    self.other_snakes = {id: snake for id, snake in
game_state['snakes'].items()
                                         if id != self.snake.player_id}

                if 'fruits' in game_state:
                    self.fruits = game_state['fruits']
            else:
                print("Disconnected from server")
                self.running = False
                break
        except Exception as e:
            print(f"Error receiving game state: {e}")
            pygame.time.delay(100)

```

What is wrong with this code???

Feel free to discuss with people around you

```

def receive_game_state(self):
    while self.running:
        try:
            data = self.client_socket.recv(4096)
            if data:
                game_state = pickle.loads(data)

                # First game state includes player ID
                if 'player_id' in game_state:
                    self.snake.player_id = game_state['player_id']
                    print(f"Assigned player ID: {self.snake.player_id}")

                # Update other snakes and fruits
                if 'snakes' in game_state:
                    self.other_snakes = {id: snake for id, snake in
game_state['snakes'].items()
                                         if id != self.snake.player_id}

                if 'fruits' in game_state:
                    self.fruits = game_state['fruits']
            else:
                print("Disconnected from server")
                self.running = False
                break
        except Exception as e:
            print(f"Error receiving game state: {e}")
            pygame.time.delay(100)

```

This is completely functional.... but is **insecure** because of pickle.

What even is a Pickle?

Pickle is a Python module that is used for serializing and de-serializing Python objects, allowing for complex objects to be put into a format that can be sent over a network. This allows the snake game to be multiplayer and communicate between the server and client.

Skipping some nitty gritty details, Pickle serialize objects into **operations** and deserializes by **executing these operations** to reconstruct the original object....



Remote Code Execution (RCE)

So what's stopping a malicious actor from, sending their own code and having pickle deserialize that. (Nothing)

It is well-documented that Pickle has this vulnerability and should **NEVER** be used on untrusted data.

Some Hope

There are some saving grace with our AI pal.....

- AI agents can help catch these security flaws
- You will have to explicitly prompt them to check for security flaws
- They can explain what the well-known flaws are

```

def receive_game_state(self):
    """Receive and validate game state from network"""
    try:
        # Read length prefix
        length_bytes = self.client_socket.recv(4)
        if not length_bytes or len(length_bytes) != 4:
            raise ConnectionError("Connection closed or corrupted data")

        data_length = struct.unpack('!I', length_bytes)[0]

        # Sanity check to prevent DoS
        if data_length > 10485760: # 10MB limit
            raise ValueError("Received data exceeds maximum allowed size")

        # Read the data in chunks
        chunks = []
        bytes_received = 0
        while bytes_received < data_length:
            chunk = self.client_socket.recv(min(4096, data_length - bytes_received))
            if not chunk:
                raise ConnectionError("Connection closed during data transfer")
            chunks.append(chunk)
            bytes_received += len(chunk)

        # Combine chunks and deserialize
        data = b''.join(chunks)
        game_state = json.loads(data.decode('utf-8'))

        # Validate received data before using
        self.validate_game_state(game_state)

        return game_state
    except (socket.error, ConnectionError, json.JSONDecodeError, struct.error) as e:
        logging.error(f"Error receiving game state: {e}")
        raise

```

This is the more secure version that uses JSON instead of pickle

Tradeoffs:

- More secure
- Slower*
- Only store basic types

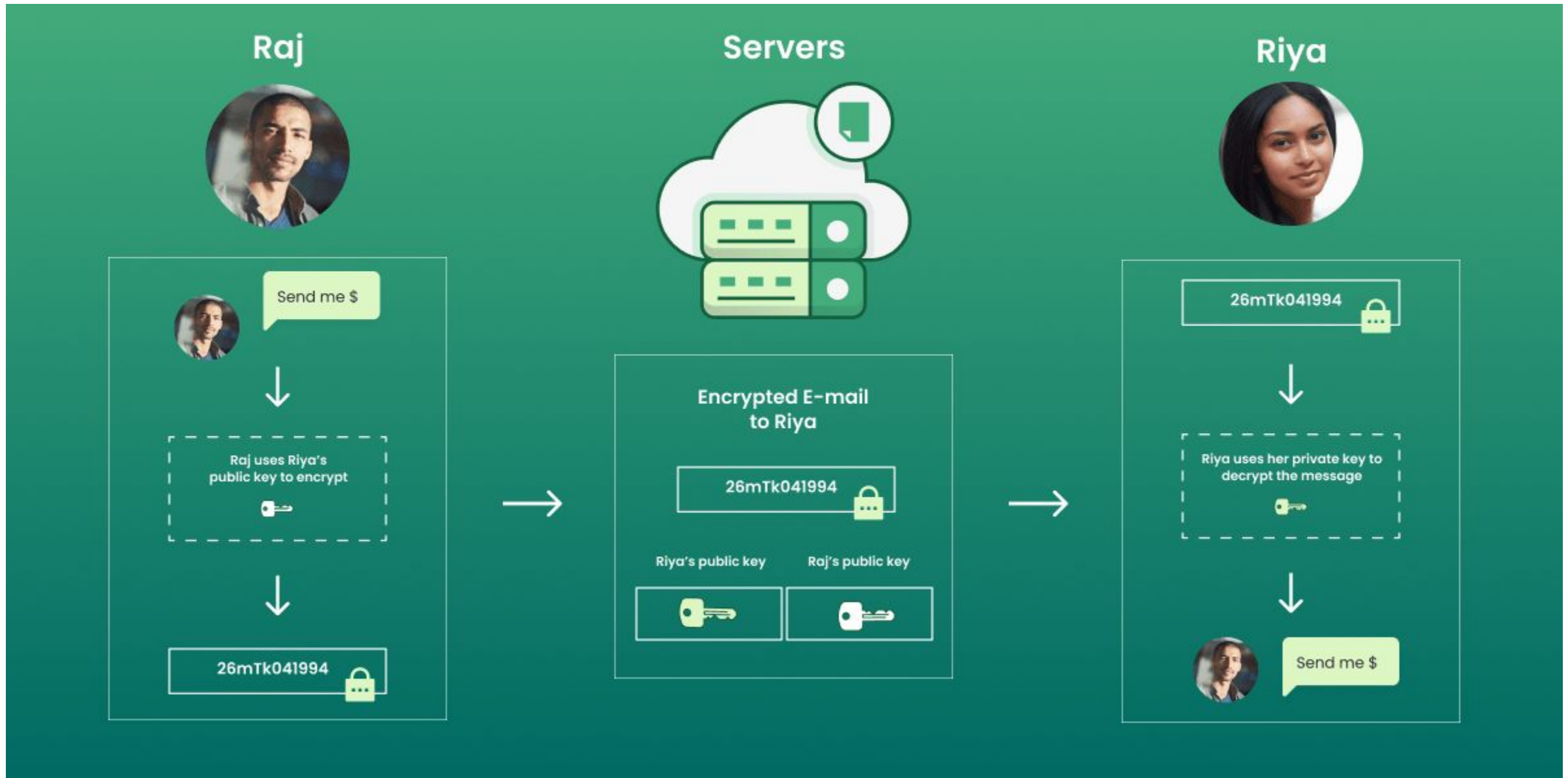
*for snake game it's negligible

One More App

- Messaging apps get a high volume of data
- Lots of moments when the message can get overseen
- Lots of points where a message can get inserted
- Messages vs. Metadata



How it works



Main Takeaways

- When dealing with data, make sure it is secure!
- Understand what your code is doing!
 - Don't blindly trust AI-generated code even if it "works"
 - Ask questions
- AI is a powerful tool that needs thoughtful thinking to use

Any Questions?